
An Adaptable Software Safety Process for Automotive Safety-Critical Systems

**Barbara J. Czerny, Joseph G. D'Ambrosio, Paravila O. Jacob,
Brian T. Murray and Padma Sundaram**
Delphi Corporation

Reprinted From: **CAE Methods for Vehicle Crashworthiness and Occupant Safety,
and Safety-Critical Systems**
(SP-1870)

ISBN 0 7680 1427-1



9 780768 014273

SAE *International*[™]

2004 SAE World Congress
Detroit, Michigan
March 8-11, 2004

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

For permission and licensing requests contact:

SAE Permissions
400 Commonwealth Drive
Warrendale, PA 15096-0001-USA
Email: permissions@sae.org
Fax: 724-772-4891
Tel: 724-772-4028



For multiple print copies contact:

SAE Customer Service
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-1615
Email: CustomerService@sae.org

ISBN 0-7680-1319-4
Copyright © 2004 SAE International

Positions and opinions advanced in this paper are those of the author(s) and not necessarily those of SAE. The author is solely responsible for the content of the paper. A process is available by which discussions will be printed with the paper if it is published in SAE Transactions.

Persons wishing to submit papers to be considered for presentation or publication by SAE should send the manuscript or a 300 word abstract of a proposed manuscript to: Secretary, Engineering Meetings Board, SAE.

Printed in USA

An Adaptable Software Safety Process for Automotive Safety-Critical Systems

Barbara J. Czerny, Joseph G. D'Ambrosio, Paravila O. Jacob,
Brian T. Murray and Padma Sundaram

Delphi Corporation

Copyright © 2004 SAE International

ABSTRACT

In this paper, we review existing software safety standards, guidelines, and other software safety documents. Common software safety elements from these documents are identified. We then describe an adaptable software safety process for automotive safety-critical systems based on these common elements. The process specifies high-level requirements and recommended methods for satisfying the requirements. In addition, we describe how the proposed process may be integrated into a proposed system safety process, and how it may be integrated with an existing software development process.

INTRODUCTION

As new, often complex, advanced automotive systems are implemented to enhance vehicle safety, performance, and comfort, system safety programs are being utilized to help eliminate potential hazards. Software is a key component in these systems. Software is increasingly controlling essential vehicle functions such as steering and braking, independently of the driver. These systems help provide significant improvements in vehicle safety, however, unexpected interactions between software and the system and its environment may lead to potentially hazardous situations.

The potential software hazards that may lead to these situations must be identified and mitigated by the system safety program. Although potential software hazards must be considered during system safety analyses, the unique aspects of software warrant that software-specific safety engineering techniques be applied. As such, following a software safety process integrated within a system safety process helps assure that best-practice software safety engineering techniques are performed, thus, providing increased confidence that the software does not create or contribute to potentially hazardous situations at the system level.

There are many software safety process standards, guidelines, and methods that exist today, including the

Motor Industry Software Reliability Association (MISRA) guidelines [1]. The existing standards and guidelines, however, do not fully address software safety in the automotive domain, nor do they address the unique aspects of the automotive domain that make an automotive-specific software safety process desirable. These unique aspects include the following:

1. Automotive suppliers work with customers from different parts of the world. Each of these may require different system and software safety standards to follow, and they may specify the specific types and levels of analyses to perform.
2. The development environment is based almost exclusively on the C programming language, unlike the military, aerospace, and nuclear power industries. Thus limiting the levels and types of analyses that can be performed.
3. Certification according to quality standards such as QS 9000 and ISO 16949 is required in order to do business. Periodic audits are required to confirm strict adherence to internally defined procedures.

These unique aspects can be addressed by a software safety process based on a set of required high-level tasks, with a corresponding set of recommended methods to implement the tasks. It is not possible to directly incorporate a rigid process standard (e.g., IEC-61508) as a required procedure without industry-wide agreement, since any divergence to meet unique project needs would result in a quality audit non-compliance. Thus, an automotive-specific software safety process must be flexible enough to accommodate different customer desires and requirements and to accommodate the varying needs during the different stages of product development, while at the same time enforcing a structured process that helps ensure software safety. The goal is to integrate best-practice elements from existing documents into a structured process that satisfies the flexibility requirements. This is achieved by a process that includes a set of high-level required tasks and recommended methods to implement the high-level tasks. The process assumes that a good underlying software development process is in place and that software safety cannot be considered

apart from system safety; the software safety process should be integrated into and compatible with established system safety and software development processes.

In the next section, we provide an overview of existing software safety documents that we reviewed in developing our proposed software safety process for the automotive domain. Next we identify and describe the common best-practice elements of a software safety process. Finally, we describe our proposed software safety process for safety-critical advanced automotive systems, discuss our experience applying the proposed process, and present our conclusions.

EXISTING SOFTWARE SAFETY DOCUMENTS

A number of software safety standards and guidelines documents and methods from various organizations exist today. This section provides a brief overview of several of these standards, guidelines, and methods.

National Aeronautics and Space Administration (NASA): NASA-STD-8719.13A provides the requirements to implement a systematic approach to software safety as an integral part of the overall system safety program [2]. The standard is intended to be applied to software that could cause or contribute to the system reaching a specific hazardous state, software that is intended to detect or take corrective action if the system reaches a specific hazardous state, and software that is intended to mitigate damage if a mishap occurs. Safety-critical software is identified during the system and subsystem safety analyses. The level of required software safety effort for a system is determined by its system category and the hazard severity level. The NASA Guidebook, NASA-GB-1740.13-96, provides more detailed information to assist in applying the standard [3].

U.S. Department of Defense: MIL-STD-882C is primarily geared toward system safety, so a detailed software safety process is not addressed [4]. It does, however, provide a software hazard risk assessment process that considers the potential hazard severity and the degree of control that the software exercises over the hardware. The software control categories are based on the level of control the software has over the hazardous function being assessed. It does not provide guidance or recommendations on the tasks or levels of analysis to perform for the determined software criticality.

Radio Technical Commission for Aeronautics (RTCA): RTCA/DO-178B provides guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements [5]. This standard provides a means of categorizing software, provides a good description of software development tasks, and links the system safety assessment process with the software development process. No specific safety tasks are detailed and the

software categories are not directly translatable to the automotive domain.

Joint Software System Safety Committee (JSSC): The JSSC Software System Safety Handbook, a Technical & Managerial Team Approach, provides management and engineering guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk [6]. The document was a joint effort between U.S. armed forces branches, with cooperation from several governmental agencies, academia, and defense industry contractors. It was intended to capture the “best practices” pertaining to software system safety. The software safety process is integrated with the system safety and software development processes. The process includes identifying generic and system safety-critical software requirements, performing appropriate software safety analyses during each stage of the software lifecycle, verifying that the software was developed in agreement with applicable standards and criteria, and developing a software safety assessment. No specific guidance is provided on determining the level of software safety effort required.

U.K. Ministry of Defense (MOD): MOD DEF STAN 00-55 emphasizes the procedures necessary for specification, design, coding, production, and in-service maintenance and modification of safety-critical software [7]. The standard identifies two categories of software: safety-related software and safety-critical software. Safety-related software is software that relates to a safety function or system and encompasses all Safety Integrity Levels (SILs). Safety-critical software is software that relates to a safety-critical function or system; this is software of the highest SIL (S4), the failure of which could cause the highest risk to human life. It provides guidance and recommendations on the requirements for developing software at the various SILs.

International Electrotechnical Commission (IEC): IEC 61508 part 3 describes the software requirements of the IEC 61508 standard and is intended to be used only after a thorough understanding of parts 1 and 2 of the standard [8]. Part 3 applies to any software forming part of or used to develop a safety-related system as described within the scope of 61508 parts 1 and 2. The level of analysis detail required is dependent on the determined software SIL. The standard includes requirements for safety lifecycle phases and activities to be applied during the design and development of the safety-related software, and requirements for software safety validation. It includes guidance and recommendations for the selection of techniques and measures to use to satisfy the determined SIL. IEC 61508 was intended to be a generic standard from which application specific standards would be developed.

Motor Industry Software Reliability Association (MISRA): MISRA compiled eight detailed reports containing information on specific issues relating to automotive software. The reports are summarized in a single

document: *Development Guidelines for Vehicle Based Software* [1]. The summary report contains information on the software lifecycle and describes three approaches for determining the integrity associated with an ECU (detailed in report 2 on Integrity [9]). Although the report does not provide an explicit process for software safety that could be directly implemented, it does provide a good overview of issues that need to be addressed when developing vehicle based software, and it contains good recommendations.

APT Research, Inc.: APT's 15 Step Process for Definition and Verification of Critical Safety Functions in Software was presented at the 2001 International System Safety Conference [10]. The 15 steps include identifying system hazards, identifying software safety functional requirements, and tailoring the safety effort to criticality. The method shows the integration of the 15 step process for software system safety into the system safety process and the software lifecycle.

Given the safety-critical nature of some advanced automotive systems, application of techniques above and beyond existing software development techniques must be considered. Suppliers of automotive systems, and automobile manufactures currently apply various safety analyses in varying degrees to systems being developed. At this point, no single software safety standard has been adopted by the automotive industry. However, there are elements common to most of the identified processes and methods, and this set of common elements can form the basis of a software safety process that provides adequate flexibility. These elements include: software safety planning, requirements analysis, architecture design analysis, detailed design analysis, code analysis, test planning, testing, test analysis, and assessments.

COMMON ELEMENTS OF A SOFTWARE SAFETY PROCESS

This section provides an overview of some typical elements of a software safety process. The software safety process proposed in this paper is an integration and adaptation of these elements.

Software Safety Planning: Software safety planning begins in the conceptual design phase of development. Inputs into this task may include the conceptual design, the System Safety Program Plan (SSPP), the Preliminary Hazard List (PHL), and the Preliminary Hazard Analysis (PHA). During this task, a plan is developed for carrying out the software safety program for the project. The Software Safety Program Plan (SWSP) identifies the software safety activities deemed necessary for the project, and is developed in conjunction with, and may be part of the System Safety Program Plan. The plan may evolve during the development process.

Software Safety Requirements Analysis: Software safety requirements analysis begins in the system and software requirements analysis phases of development. Inputs

into this task may include the system safety requirements, the system safety concept, the PHA, and the software requirements. The purpose of this task is to identify safety-critical software requirements, to help ensure that the decomposition of the system level safety requirements to the software safety requirements is complete and consistent, and to provide safety-related recommendations to the design and testing phases. Safety-critical software requirements are identified during the system PHA to eliminate, mitigate, or control hazards related to software. Software safety requirements may also stem from government regulations, customer requirements, or internal requirements. Information for eliminating, mitigating, or controlling hazards related to software, and safety-related design and testing recommendations are passed on to the architecture design phase. A matrix identifying software safety requirements may be initiated to track the requirements throughout the development process.

Software Safety Architecture Design Analysis: The software safety architecture design analysis task begins in the system and software architecture design phases. Inputs into this task may include the system architecture design, the system hazard analyses outputs (e.g., the PHA and safety concept), the safety-related design and testing recommendations from the software safety requirements analysis task, the software architecture design, the software safety requirements, and software criticality and tailoring guidelines. The PHA and software safety requirements are reviewed to determine if additional hazards related to software can be eliminated, mitigated, or controlled, and additional information on eliminating, mitigating, or controlling the hazards is passed on to the detailed design phase. Software components and functions are identified in the software architecture design phase. The software components and functions that implement the software safety requirements or that affect the output of the software safety requirements are identified as safety-critical. A software criticality level may be determined for the safety-critical software components and functions. Software criticality levels indicate the potential level of risk associated with different software components. The correctness and completeness of the software architecture design as it is related to the software safety requirements and the safety-related design recommendations is analyzed to help ensure that the design satisfies the software safety requirements. Safety-related recommendations for the detailed design and test procedures are provided, and test coverage of the software safety requirements is verified.

Software Safety Effort Tailoring: Tailoring the software safety effort is an umbrella task that begins when the safety-critical software components and functions are identified and assigned criticality levels. This task is relevant to all software safety tasks. Inputs into this task include the system safety requirements, system safety hazard analysis outputs, software safety requirements, and the software safety architecture and detailed design analysis outputs. Appropriate levels and types of analyses are identified based on the determined

criticality level from the software safety architecture design analysis. Information on suggested levels and types of analyses, testing, and verification and validation for the identified criticality levels may be obtained from tailoring guidelines tables if they exist. The output from this task is the tailoring recommendations. Criticality levels and tailoring recommendations may be tracked in the software safety requirements matrix.

Software Safety Detailed Design Analysis: Software safety detailed design analysis begins in the software detailed design phase. Inputs into this task may include the system hazard analyses (e.g., the detailed hazard analysis and hazard control specifications), the system and software detailed designs, the software safety requirements, software architecture design analysis output, safety-related detailed design recommendations, and tailoring recommendations. The identified safety-critical software components and functions that implement the software safety requirements are refined to the unit level software components and functions. The system and software detailed designs are analyzed to help ensure that the software detailed design satisfies the software safety requirements. Subsystem interfaces may be analyzed to identify potential hazards related to interfacing subsystems, such as hazardous interface failure modes and data errors. Test coverage of software safety requirements is verified, and safety-related recommendations for the software implementation are provided. Software safety detailed design analysis continues during a portion of the software implementation and unit testing phases. Outputs from this task may include the identified safety-critical unit level software components and functions, the identified subsystem interfacing hazards, and safety-related software implementation and test coverage recommendations. Any identified subsystem interface hazards are input back to the relevant system hazard analyses.

Software Safety Code Analysis: This task begins in the software implementation and unit-testing phase. Inputs into this task may include the system hazard analyses outputs (i.e., detailed hazard analyses outputs), software safety requirements, software detailed design, software safety detailed design analysis output, safety-related software implementation recommendations, software implementation, and the tailoring recommendations. The completeness and correctness of the code as related to the software safety requirements, software detailed design, and safety-related software implementation recommendations is analyzed to help ensure that the software safety requirements are satisfied in the implementation. The analysis may check for potentially unsafe states that may be caused by I/O timing, out-of-sequence events, adverse environments, hardware failure sensitivities, failure of events, etc. Test coverage of the software safety requirements is analyzed. This information may be tracked in the software safety requirements matrix. A software implementation report describing the types of analyses performed and the results of the analyses, and software implementation test coverage recommendations may be written.

Software Safety Test Planning: Software safety test planning begins in the software architecture design phase and continues through the software integration and acceptance testing phase. Inputs into this task may include system hazard analyses outputs (i.e., PHA, safety concept, detailed hazard analyses, and hazard control specifications), system safety test plans and test procedures, software safety requirements, software test plans and test procedures, and tailoring recommendations. During this task, appropriate software safety tests that address all identified potential hazards related to or affected by the software are incorporated into the software safety test plan. The software safety test plan is developed to help ensure that all identified safety requirements related to or affected by software will be adequately tested. Test procedures should include both nominal and off-nominal conditions. System safety test plans and test procedures related to software are examined and additional system safety test plans and procedures are developed as required. The software safety test plan may be part of the software test plan. Testing and verification requirements may be included in the software safety requirements matrix. This facilitates the tracking and verification process to help ensure that the software safety requirements are satisfied and appropriately tested and verified.

Software Safety Testing and Test Analysis: These tasks begin in the software implementation and unit testing phase. Inputs into the software safety testing task include the system and software safety test plans and procedures. Inputs into the software safety test analysis task may include the software safety requirements, system safety program plan, software safety program plan, system and software safety test plans and procedures, and the system and software safety test results. Planned software safety tests are performed, and test results are reviewed to help ensure that safety requirements have been satisfied. This helps ensure that the identified potential hazards have been eliminated or controlled to an acceptable level of risk according to the system and software safety program plans. Any software problems identified are corrected, and follow-up tests are defined to verify that the identified software problems were corrected and that no additional problems were introduced into the system. The appropriate systems people are notified if a system corrective action is required.

Verify Software Developed in Agreement With Applicable Standards / Guidelines: This task begins in the software architecture design phase and continues through to the latter stages of development. Inputs into this task may include the system safety program plan, software safety program plan, and applicable standards / guidelines. The standards / guidelines may come from the customer, government regulations, or they may be internal standards / guidelines. The goal of this task is to help ensure that all applicable safety-related standards and guidelines identified in the program plans have been adhered to in the development of the safety-critical software components and functions. Any discrepancies in adhering to the standards / guidelines are identified

and recommendations are made for alleviating the discrepancies.

Software Safety Assessments and the Software Safety Case: Software safety assessments begin in the early stages of development, and a software safety assessment is completed at each major gate review during the project development process. This assessment describes the current state of safety in the software being developed. The assessment indicates any known issues and what will be done to resolve them. Issues from previous safety assessments that have been closed are identified and marked as closed. The software safety case provides supporting documentation and justification as to why the developers believe the software as developed is safe. This documentation is developed from the final software safety assessment. All open issues from the final software safety assessment should be closed. If issues remain open, justification must be provided as to why the open issues are acceptable. Any residual risk associated with software that remains in the system and that has been determined acceptable, is justified in the software safety case. Inputs into these tasks include outputs from all system and software safety tasks, and previous system and software safety assessments. The software safety assessments and the software safety case may be part of the system safety assessments / case respectively.

In the next section we describe the proposed software safety process that combines the elements described in this section.

A SOFTWARE SAFETY PROCESS FOR SAFETY-CRITICAL ADVANCED AUTOMOTIVE SYSTEMS

The software safety process described here contains elements from several existing standards, guidelines, and methods. These elements have been tailored and combined in an effort to make them applicable to the automotive domain; specifically to safety-critical, advanced automotive systems. The main sources used are the JSSC Software System Safety Handbook, NASA-STD-8719.13A, and APT Research, Inc.'s 15 step process (Figure 1). The other documents contain much useful information as well and are used to provide additional information.

In addition to the previously discussed criteria that need to be satisfied for the automotive industry, Delphi has certain internal requirements that need to be satisfied. The internal requirements are that the software safety process:

- must be integrated into and compatible with the proposed system safety process for safety-critical advanced automotive systems,
- must be integrated into and compatible with established software development processes and compatible with established coding standards, and

- must be adaptable to different projects in different stages of the development process.

To support different stages of development, the levels and types of analyses chosen are typically more rigorous for a production design than for a prototype design.

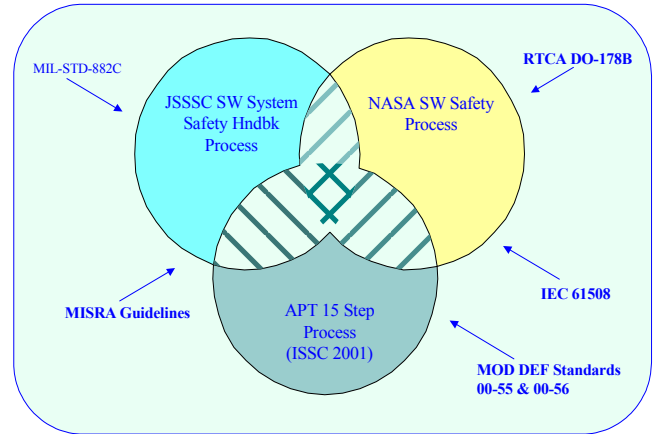


Figure 1: Integration of software safety documents.

SOFTWARE SAFETY PROCESS ARCHITECTURE

Figure 2 shows the software safety process architecture being developed within Delphi for safety-critical, advanced automotive systems. Existing software safety process documents were reviewed and best-practice elements were identified. These best-practice elements form the foundation of our proposed software safety process architecture. The process consists of a software safety procedure that contains a set of high-level required tasks, and a set of recommended methods to implement the high-level tasks. The software safety procedure high-level tasks are generic enough to be applicable to all safety-critical systems. The set of recommended methods to implement the high-level tasks may be tailored for specific projects in different stages of development and for different customer requirements.

Figure 3 shows the common best-practice elements that make up the foundation of our proposed software safety process. The diagram shows the software safety tasks, general process flow of the tasks, the relationships between tasks, and the duration of the tasks. Task boxes that overlap vertically, may be carried out simultaneously. For example, software safety test planning begins during the software safety architecture design analysis and continues into software safety testing and test analysis. Software safety assessments begin during the software requirements analysis task and continue throughout the process, ending in the final software safety case. Not all of the process flow options are shown since the figure would become too cluttered. Only the main flows are shown; the same will be true for other process description figures that follow.

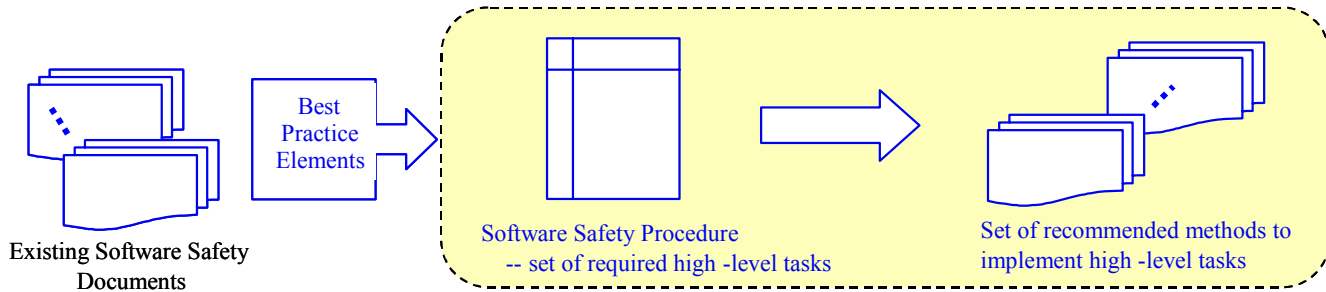


Figure 2: Software safety process architecture.

goal is to ultimately incorporate the software safety process aspects directly into the software development procedures. The phase two integration will be a tight integration of the two processes, where the actual high-level required tasks from the software safety process will be directly incorporated into the software development process procedures (Figure 5). For example, the software safety planning task will be directly integrated into the software planning procedure. Within the software planning procedure will be a requirement that if the system being developed is safety-critical, then software safety planning must be performed. The same requirement will be present in the other software development procedures for their corresponding software safety tasks.

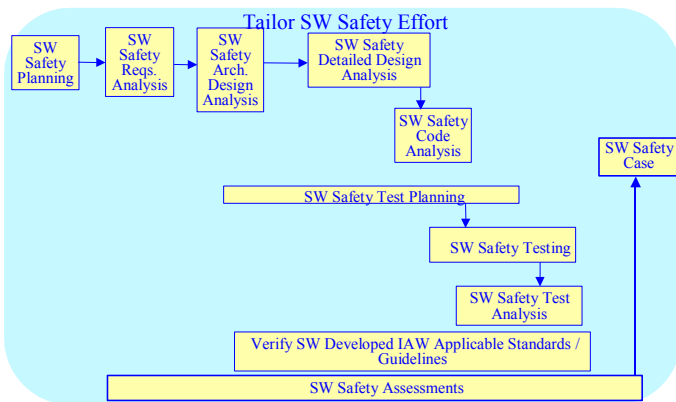


Figure 3: Software safety tasks and relationships.

INTEGRATION WITH SOFTWARE DEVELOPMENT PROCESS

The software safety process will be used in conjunction with an existing software development process within Delphi. The software safety process will be integrated into the software development process in two phases. The initial phase one integration (Figure 4) will be a loose integration of the two processes and consists of identifying the procedures required to satisfy the software development process and linking their inputs and outputs with the software safety process inputs and outputs. Phase one integration will be an evaluation and modification stage. The software safety process will be applied on various safety-critical advanced automotive systems and adapted based on feedback from the application of the process. Once we are fairly confident that the process works well for various applications, we will move to the phase two integration with the existing software development process.

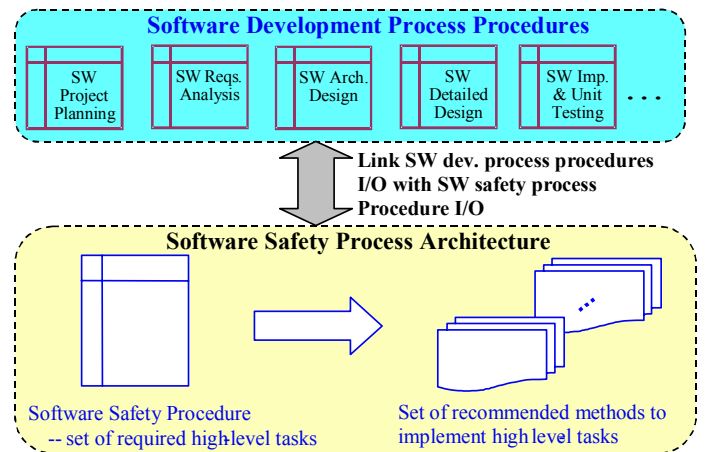


Figure 4: Phase one integration with software development process.

Since any software safety process developed is closely integrated with the software development process, the

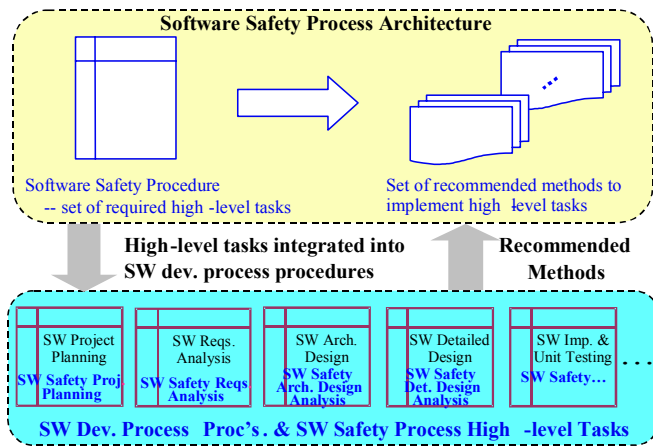


Figure 5: Phase two integration with software development process.

Figure 6 shows a phase one, loose integration view of the tasks of the software development process and the tasks of the software safety process and the relationships between the tasks. The tasks in the upper shaded area are tasks in the software development process. Tasks shown outside of boxes in the shaded areas are umbrella tasks and are performed throughout the corresponding process (i.e., SW Requirements Traceability, SW Configuration Management, etc. are applicable throughout the software development process, and the Tailor SW Safety Effort task is applicable throughout the software safety process). As previously stated, the lengths of the boxes in the figure show the duration of the tasks and the vertical relation of boxes shows the various tasks that may overlap. The software development process tasks are outside the scope of this paper and will not be discussed in detail. The tasks that occur during each of the software safety process steps are the same as those described under the Elements of a Software Safety Process section and will not be further described here.

INTEGRATION WITH SYSTEM SAFETY PROCESS

Since software safety cannot be considered apart from system safety, the software safety process should be part of an established system safety process as well. Figure 7 shows the relationships between all three processes, the proposed system safety process for safety-critical advanced automotive systems, the established software development process, and the proposed software safety process. The figure shows the phase one loose integration of the three processes. Software safety requirements are initially obtained from the system safety PHA when potential hazards are identified that may be related to software. Additional safety requirements may be identified during later system and software process steps.

Results affecting the software safety process obtained from system safety analyses are communicated to the software developers during the appropriate stage of

development. For example, the appropriate levels and types of analyses for the high-level software safety process tasks are determined based on identified software criticality levels that follow from the results of the system safety analyses, the project's development phase, and customer requirements. Likewise, information obtained during the software safety process that affects the system safety analyses results is communicated to the appropriate system developers.

Software and system safety assessments start during the early stages of both the system and software safety processes, and continue through the development of the final system and software safety case. A system and software safety assessment is presented at each major gate review during the development process, with the final safety assessment forming the basis for the safety case.

In addition to defining the software safety tasks and the relationships between the tasks of the software development, software safety, and system safety processes, we have also identified the inputs and outputs that may be acted on, generated by, and used between the processes. Figure 8 shows possible inputs and outputs that may be acted on and generated by the software safety process. For example, outputs generated by the SW Safety Requirements Analysis task are the software safety requirements, and design and testing recommendations. The software safety requirements become inputs into and may be acted on by the SW Safety Architecture Design Analysis, Detailed Design Analysis, Code Analysis, and so on. Similar diagrams exist for the inputs and outputs that may be exchanged between the system safety and software safety processes, and between the software development and software safety processes.

DISCUSSION

The proposed software safety process has been applied to projects in different stages of development (e.g., prototype and production) [11]. Application of the proposed process to these projects confirmed the need for flexibility and adaptability of the process. Prototype projects are generally attempting to prove a concept and are characterized by rapidly changing requirements. A prototype vehicle is typically built, and operating restrictions can be placed on the use of the prototype vehicle. In these cases, it is not necessary that a safety case and associated validation testing be fully completed. However, the development team may decide that a basic understanding of software safety issues is important if these issues will have a significant impact on the later phases of product development. For a production project, a product is developed for a specific customer application and put into production. This level of development requires a more rigorous application of established safety processes.

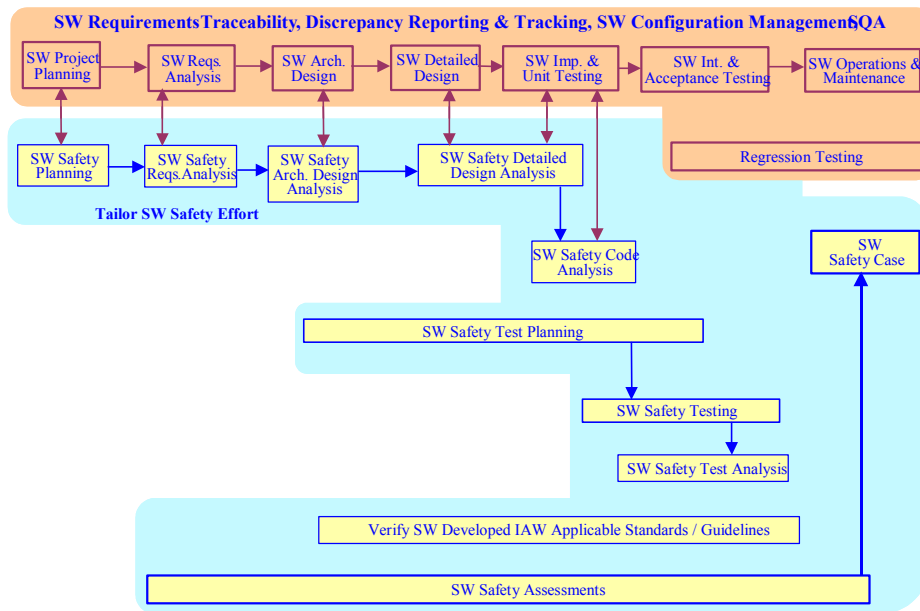


Figure 6: SW development and SW safety process task relationships.

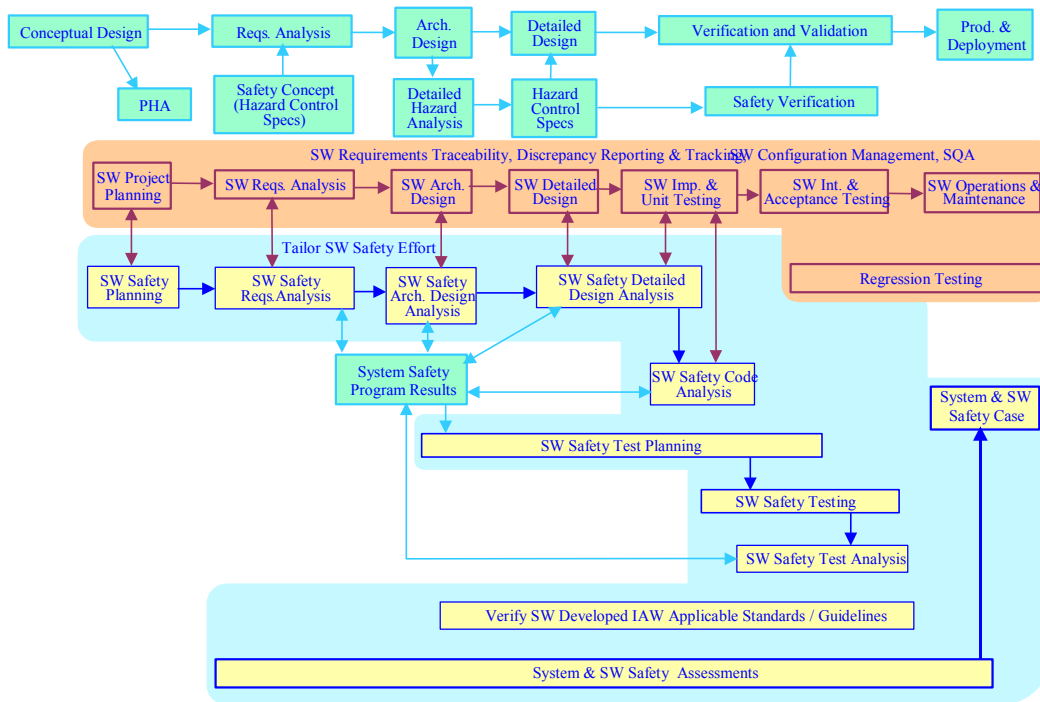


Figure 7: System safety, SW development, and SW safety process task relationships.

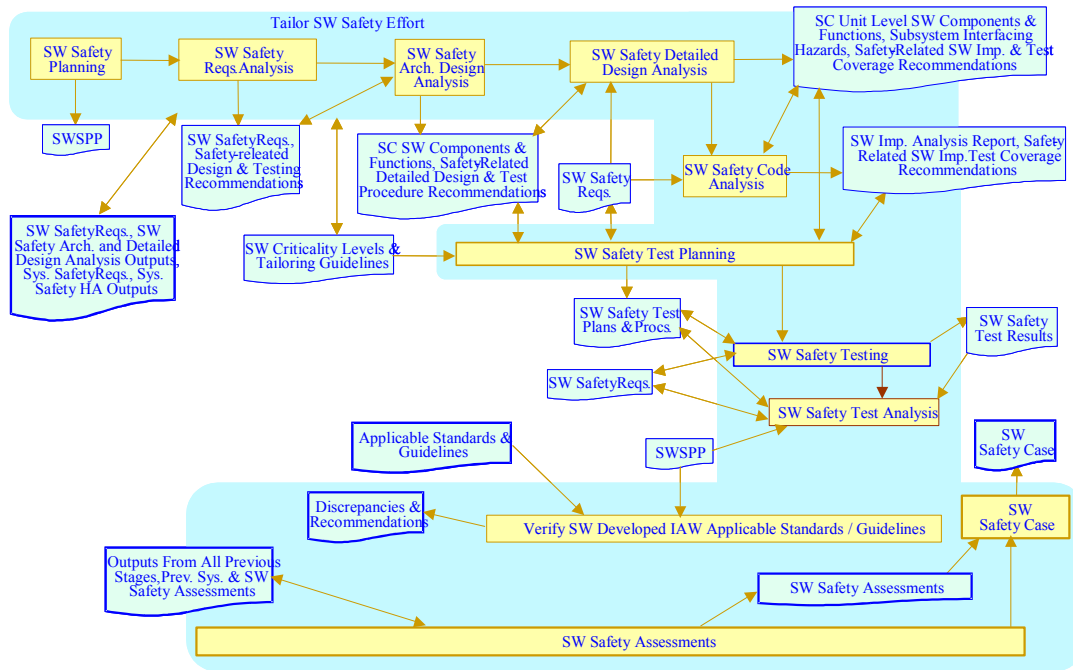


Figure 8: Software safety process inputs and outputs.

For the projects the proposed process was applied to, the levels and types of analyses selected for the various software safety process tasks varied depending on the project's stage of development, however, there was some overlap in the selected levels and types of analyses chosen. For the selected methods that overlapped, some proved beneficial across both types of projects, while others were beneficial only on production projects. For example, the more detailed level analysis applied to a prototype project during the software safety detailed design analysis proved to be too time consuming for this application and was never fully completed. In contrast, the software safety detailed design analysis applied on a production project proved effective in analyzing the integrity of the software design and its potential impact on system hazards. In all cases, the need for a software safety activity was confirmed, however, detailed analysis was deemed most suitable for production projects. These and other lessons learned have been incorporated into the process definition.

Overall, the process appears to be efficient and to meet Delphi's needs. It provides a structured software safety process that fosters consistent application of best-practice software safety engineering techniques to safety-critical advanced automotive systems. It is adaptable so that Delphi can satisfy different customer desires and requirements, and it is flexible enough to work in different stages of product development.

CONCLUSION

Software safety is important for safety-critical advanced automotive systems, especially since software is

increasingly controlling essential vehicle functions such as steering and braking, independently of the driver. A software safety process used in conjunction with a system safety process can help ensure that appropriate software-specific safety engineering techniques are applied. As discussed in the paper, it is not possible for Delphi to adopt an existing rigid software safety process. A software safety process feasible for Delphi must be adaptable to different customers and to projects in different stages of development.

In this paper we reviewed existing standards, guidelines, and methods for software safety. In addition, we described some common elements of a software safety process derived from the reviewed documents, and then presented how these elements are integrated to form the proposed automotive, software safety process. The process is based on a set of required high-level tasks with a corresponding set of recommended methods for implementing the tasks based on the determined software criticality level. The recommended methods are adaptable to the specific needs of individual projects. We demonstrated how the software safety process may be integrated with the existing software development process, and how it may be integrated with our proposed system safety process.

Initial applications of the process have been positive. As we move forward, the set of recommended methods will be revised appropriately based on the results of future applications. When we have gained experience with the overall process and its generic applicability, we will be in a position to decide if the process should become a required procedure.

REFERENCES

1. MISRA. *Development Guidelines for Vehicle Based Software*. November 1994.
2. NASA. *Software Safety: NASA Technical Standard NASA-STD-8719.13A*. September 1987.
3. NASA. *Guidebook for Safety Critical Software NASA-GB-1740.13-96*.
4. Department of Defense. *System Safety Program Requirements, MIL-STD-882C*. 1984.
5. RTCA. *SW Considerations in Airborne Systems and Equipment Certification RTCA/DO-178B*. 1994.
6. D. Alberico, J. Bozarth, M. Brown, et. al. *JSSC Software System Safety Handbook; A Technical and Managerial Team Approach*. December 1999.
7. Ministry of Defence. *Requirements for Safety Related Software in Defence Equipment; MOD DEF STD 00-55.; Part 1: Requirments; Part 2: Guidance*. August 1997.
8. IEC. *International Standard; Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems – IEC 61508-3; Part 3: Software Requirements*. 1998.
9. MISRA. *Report 2; Integrity*. February 1995.
10. H. D. Kuettnner, Jr. and P. R. Owen, "Definition and Verification of Critical Safety Functions in Software", in *Proceedings of the International System Safety Conference (ISSC) 2001*. System Safety Society, Unionville, Virginia 2001. pp. 337-346.
11. B. J. Czerny, J. G. D'Ambrosio, Paravila O. Jacob, et. al. *A Software Safety Process for Safety-Critical Advanced Automotive Systems*, Proceedings of The International System Safety Conference, August 2003.

CONTACT

Barbara J. Czerny, Ph.D., Sr. System Safety Engineer, Delphi, Corp., 12501 E. Grand River, MC 483-3DB-210, Brighton, MI, 48116-8326; Phone: (810-494-5894), Fax: (810) 494-4689, email: barbara.j.czerny@delphi.com